* ÁRVORES GERADORAS MÍNIMAS * (Cormen, cap. 23)

0. OBSERVAÇÃO: veja a nota de aula com as definições básicas sobre grafos.

1. DEFINIÇÃO DO PROBLEMA: Dado um grafo não-direcionado e conexo G = (V,E), com arestas ponderadas por uma função w : E -> R, desejamos encontrar um subgrafo gerador T de G que seja uma árvore e tenha o menor peso possível.

2. ALGORITMO DE KRUSKAL, em alto nível:

```
============================================================
Algoritmo: kruskal_alto_niv
Entrada: "G = (V,E)" e "w : E -> R", como descrito acima.
Saída: um conjunto A ⊆ E.
------------------------------------------------------------
1. A := ∅
2. Ordene as arestas de G em ordem crescente de peso
3. ENQUANTO |A| < |V|-1
4. | e := a aresta mais leve ainda não considerada
5. | SE G[A ∪ {e}] for acíclico
6. | | A := A ∪ {e}
7. RETORNE A.
============================================================
```

3. DEFINIÇÃO: se (G,w) é uma entrada para o problema em questão, então uma PRÉ-AGM de G é um conjunto de arestas A tal que, para alguma AGM T = (V,F) de G, A ⊆ F.

4. LEMA: "A é uma pré-AGM de G" é um invariante do laço do algoritmo acima.

```
================================================================================
```

PROVA:

De fato, logo antes do laço, A é o conjunto vazio, que é subconjunto do conjunto de arestas de qualquer AGM de G, e portanto A é pré-AGM de G.

Nós mostraremos agora que a afirmação "A é uma pré-AGM de G" é mantida verdadeira em cada iteração do laço. De fato, consideremos uma iteração qualquer do laço, e, como A é pré-AGM de G no início da iteração, seja T = (V,F) uma AGM de G tal que A ⊆ F. São então dois os casos possíveis:

  * CASO 1: A ⊆ F ao fim da iteração. Nesse caso, A obviamente é pré-AGM de G, CQD.

  * CASO 2: A ⊄ F ao fim da iteração. Nesse caso, nós mostraremos que, ao fim da iteração, A ⊆ H, para algum H tal que U = (V,H) é AGM de G. De fato, seja B o valor de A no início da iteração. Logo, B ⊆ F e B ∪ {e} ⊄ F. Além disso, sejam "u" e "v" as extremidades de "e". Como e ∉ F, então existe em T um caminho "p" de "u" a "v" diferente de <u,v>, e portanto que possui pelo menos duas arestas. Além disso, existe em "p" uma aresta e' = {a,b} que não pertence a B: se esse não fosse o caso, então todas as arestas de "p" pertenceriam a B, e, nesse caso, G[B ∪ {e}] seria cíclico, o que não é o caso, pois, na iteração em questão, o algoritmo inseriu "e" em "A".

Nós mostraremos agora que w(e') >= w(e):

| De fato, suponha por absurdo que w(e') < w(e). Logo, o algoritmo considera
| a aresta "e'" para inserção antes de "e". Nós mostraremos agora que,
| quando o algoritmo considera a aresta e', ele a insere no conjunto A:
|
| | De fato, suponha por absurdo que o algoritmo não insere e' em A. Isso
| | significa que, na iteração em que e' é considerada p/ inserção, as suas
| | extremidades "a" e "b" já pertencem à mesma componente conexa em G[A],
| | isto é, existe em G[A] um caminho de "a" a "b" que não utiliza a aresta
| | e'. Assim, como o conjunto A não diminui durante a execução do
| | algoritmo, então, no início da iteração posterior do algoritmo em que
| | ele considera a aresta "e" para inserção, continua havendo em G[A] um
| | caminho de "a" a "b" que não utiliza a aresta e'. Logo, como A ⊆ F vale
| | no início dessa iteração, então o caminho em questão existe também em T.
| | Entretanto, e' ∈ F, e portanto existe em T um ciclo envolvendo os
| | vértices "a" e "b", um absurdo, pois T é uma árvore.
|
| Logo, no início da iteração em que a aresta "e" é considerada pelo
| algoritmo, vale e' ∈ A, um absurdo, pois nós supuséramos que e' ∉ B, ou
| seja, que e' ∉ A no início da iteração em questão.

Assim sendo, considere o grafo U = (V,H) tal que H = (F \ {e'}) ∪ {e}. Como w(e') >= w(e), então w(U) <= w(T). Além disso, observe que U é uma árvore:

| De fato, como a remoção de uma aresta de uma árvore sempre leva a um grafo
| com exatamente duas componentes conexas, então grafo T' = (V, F \ {e'})
| possui exatamente duas componentes conexas. Além disso, "u" e "v"
| pertencem a componentes conexas distintas em T', pois, se assim não fosse,
| existiriam em T dois caminhos diferentes de "u" para "v", um passando pela
| aresta e' e outro não passando por ela, o que seria um absurdo, pois, numa
| árvore, existe exatamente um caminho ligando quaisquer dois vértices.
| Assim sendo, o grafo U é necessariamente conexo, pois U é o grafo que
| resulta da adição da aresta e = {u,v} ao grafo T'. Além disso, U é
| acíclico, pois não T' é acíclico e não existe caminho de "u" a "v" em T'.
| Logo, U é uma árvore, como desejado.

Logo, U = (V,H) é uma AGM de G, e B ∪ {e} ⊆ H, CQD.

Nós concluímos então que "A é uma pré-AGM de G" é um invariante do laço do algoritmo, CQD.

==========

5. EXERCÍCIOS: o tempo de aula da nossa disciplina é relativamente curto para a exploração de todos os aspectos do algoritmo de Kruskal em detalhes. Os exercícios a seguir sugerem a complementação do que foi abordado em sala:

a) Utilize o lema acima para argumentar que o algoritmo de Kruskal é correto.

b) O algoritmo de Kruskal pode ser implementado de forma a executar em tempo O(m*lg m) = O(m*lg n), onde n = |V| e m = |E|. Utilize a implementação de conjuntos disjuntos por listas encadeadas (Cormen, §21.2) para obter uma implementação de Kruskal que executa nessa complexidade de tempo.